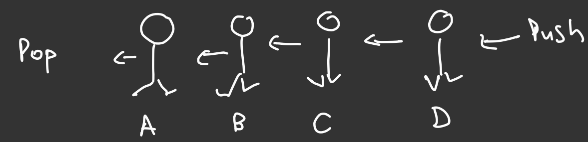
"Heaps" / Priority Queue

↓

Data Structure that allows to build a "Priority Queue"

Queue

(FIFO)



Priority Queue

by "Age"

E – 15

$\boxed{A - 63}$

$\boxed{B - 28}$

$\boxed{C - 34}$

$\boxed{D - 19}$

E –

~~A~~ – B – ~~C~~ – D

1	$\boxed{A}$	63
2	$\boxed{C}$	34
3	$\boxed{B}$	28
4	$\boxed{D}$	19
5	E	15

Pop operation
↓
decided by biggest no

Q 1000 appeared for exam Top-3 students

Top-K

→ Rank 1
→ Rank 2
→ Rank 3
{ : }

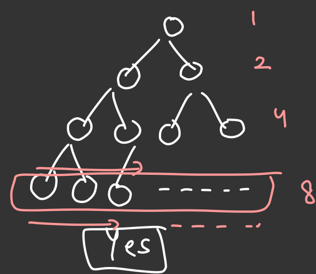
Remove Top-3 from the queue

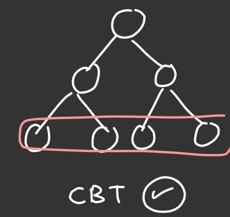
(faster than doing sorting)

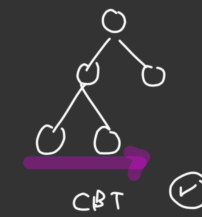
• Heap Data Structure

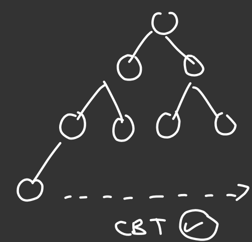
① Complete Binary Tree (Structure)

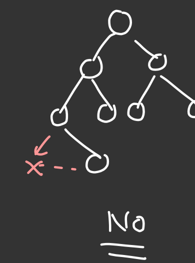
BT in which all levels are completely filled except last level which may be partially filled but the filling must be in left to Right order





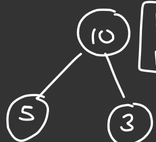






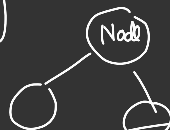
2) Heap order Property

Max Heap or **Min Heap**



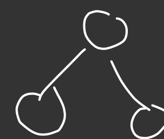
Max Heap

$$\left[\begin{array}{l} 10>5 \\ 10>3 \end{array}\right]$$

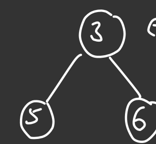


Node.data $\geq$ Children.data

at every node



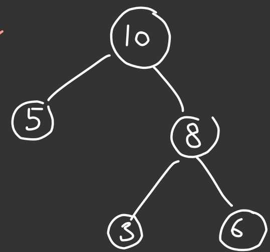
Node.data $\leq$ children data



$3<5$

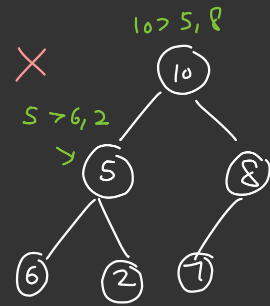
$3<6$

Examples of Max Heap –



[Heap Property ✓
CBT ×]

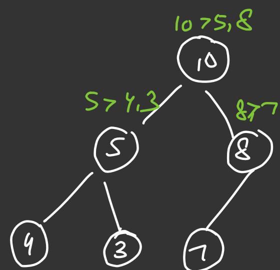
Not a heap



CBT ✓

Max Heap ×

Not a heap

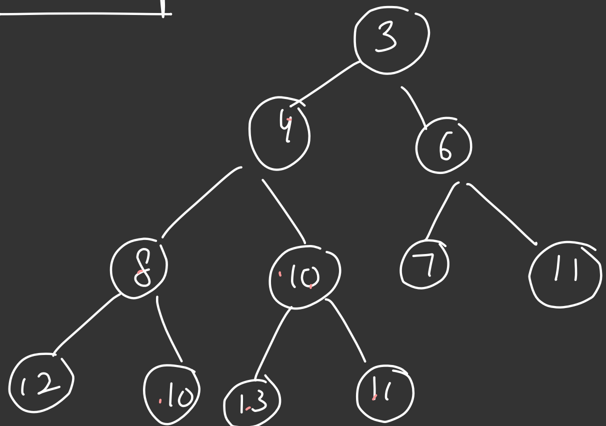


→ CBT
→ Heap max]

$\boxed{\text{Max Heap}}$

getMax() in O(1)

Min Heap



CBT ✓

Min Heap ✓

Advantage getMin() → Root

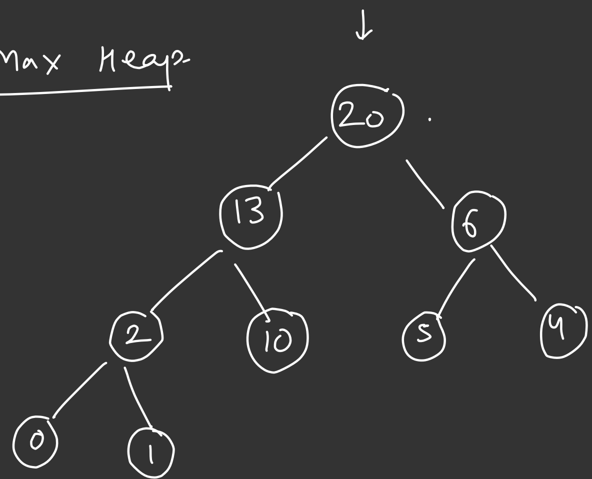
⇒ O(1) time

A binary Tree is a heap if it follows

① → CBT

② → Min Heap Prop / Max Heap Prop.

Max Heap



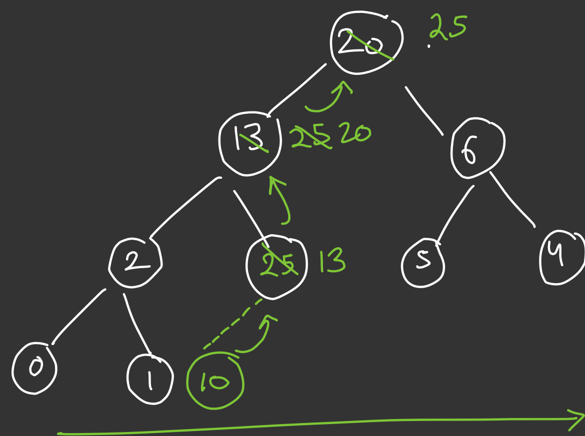
Marks

13, 6, 20, 2, 10, 5, 4, 0, 1

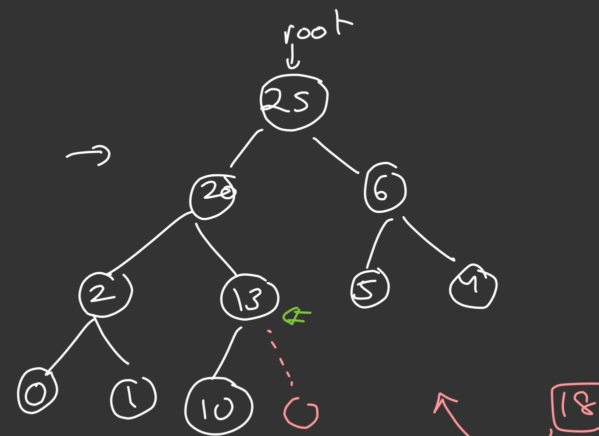
Insert - 25

Insert -

data $\boxed{25}$



→



$\boxed{\text{CBT}}$ ✓

$\boxed{\text{Max Heap}}$ ✓

$\boxed{\text{Heap}}$

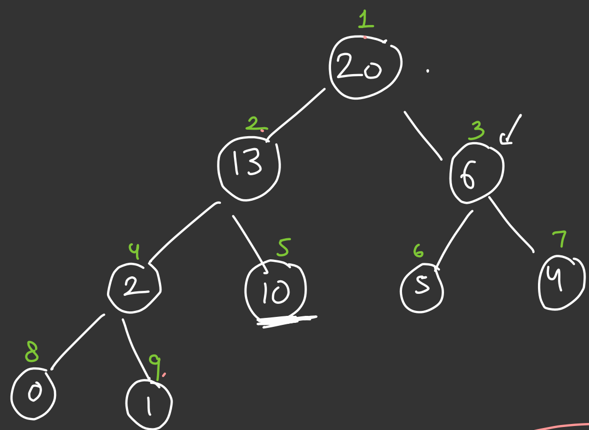
Height - of CBT ⇒ $O(\text{Log } N)$

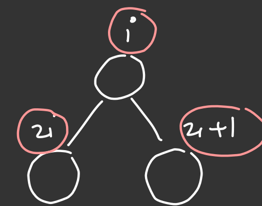
Insertion TC : First null → ~~Traversal~~ ⇒ $O(N)$ + Move Up $O(\text{Log } N)$

Avoid at all cost

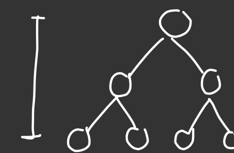
kills the adv.

Solution   Visualise Tree but build a array



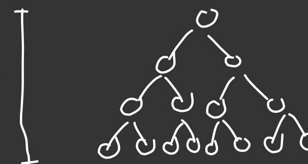


Since it is a $\boxed{\text{CBT}}$



$N = 7$

$\log(7+1) = 3$

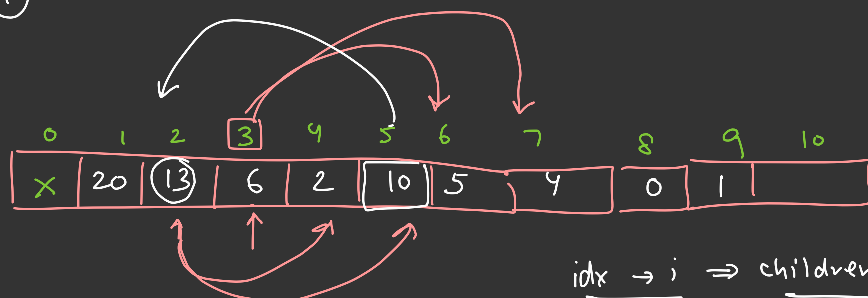


$N = 15$

$\log(15+1) = 4$

$\boxed{H = \log N}$

array ⇒



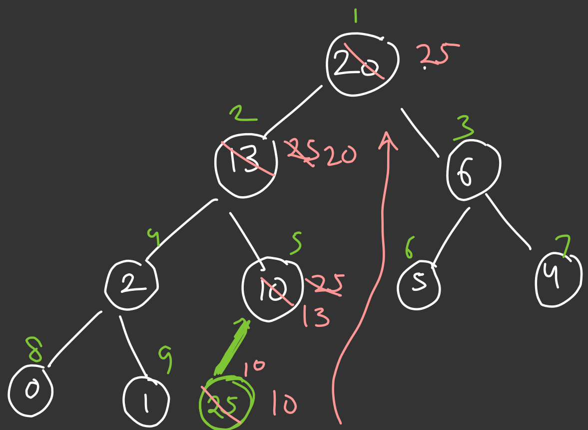
also stores the tree hierarchy

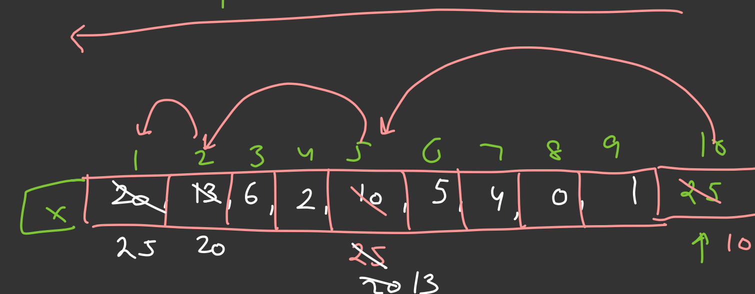
idx → i ⇒ children → $2i, 2i+1$

i ⇒ parent $i/2$

CBT

Use an array to build a CBT (Heap.

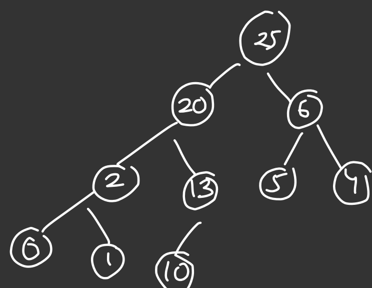




$i \Rightarrow N$

$idx = \boxed{10}$

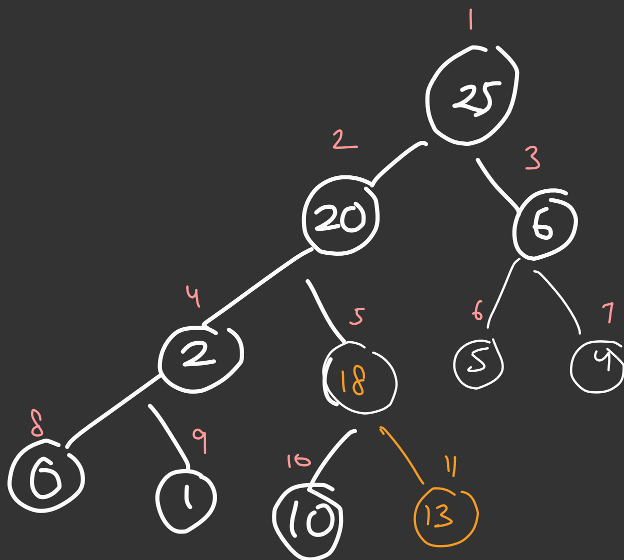
$\rightarrow$ arr[idx] = 25;



[25, 20, 6, 2, 13, 5, 4, 0, 1, 10]

$$\underbrace{N \rightarrow N/2 \rightarrow N/4 \rightarrow \ldots 1}_{\log N \text{ steps}}$$

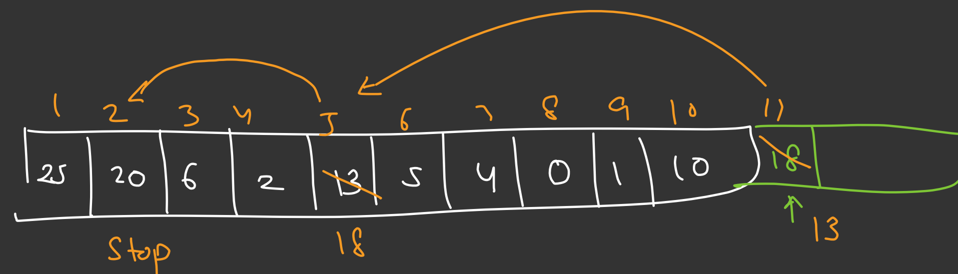
Insert = $O(\log N)$ in heap



N = 10

ins → $\boxed{18}$

[- CBT
- Max Heap.]



Heap

~~1~~ ↳ Insert (data) ✓

~~2~~ ↳ getMax () ✓

3 ↳ removeMax () [...]

10 Mins

```
class Heap {
    int arr[]
    Heap( maxSize){
        arr = new int[maxSize].
    }
    :
    int getMax() {
```

10 min

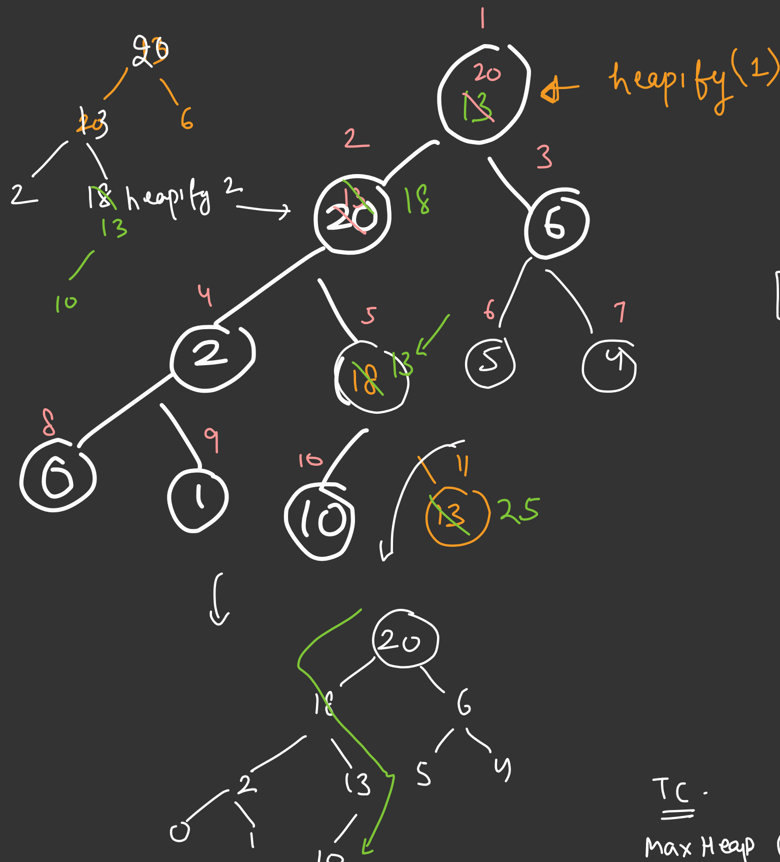
exercise

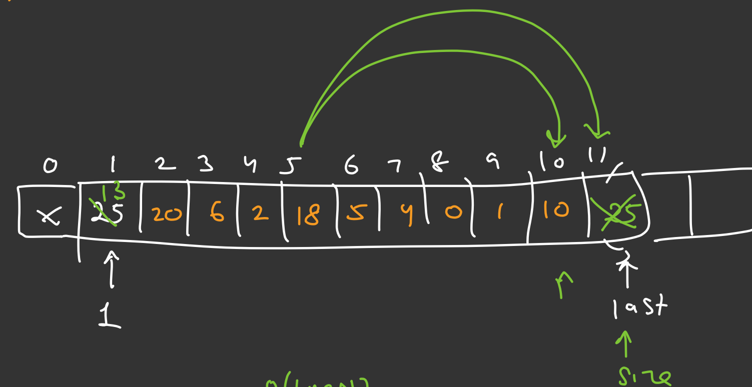
10 03

```
}
void insert ( ) {


}

}
```





O(logN)

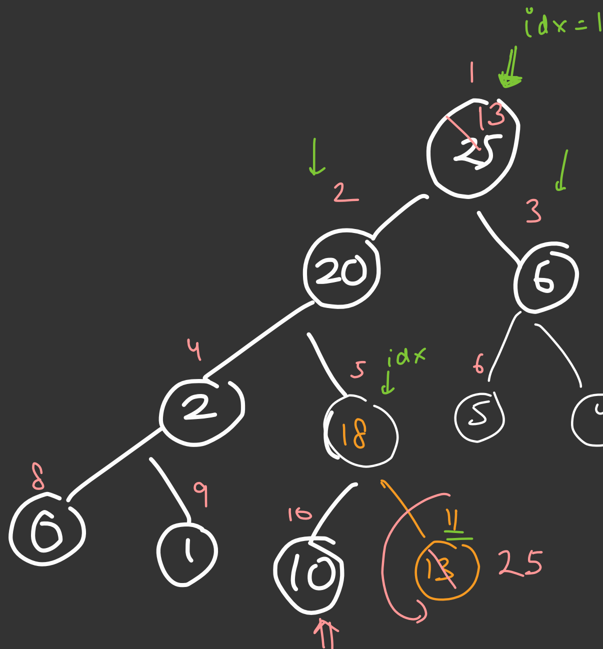
removeMax()

- → swap(a[1], a[last])
- → size = size - 1
- heapify ↓ = O(log N)

TC.

Max Heap ✓

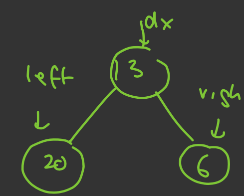
Next largest ⇒ 20



$left = 2 \cdot idx$

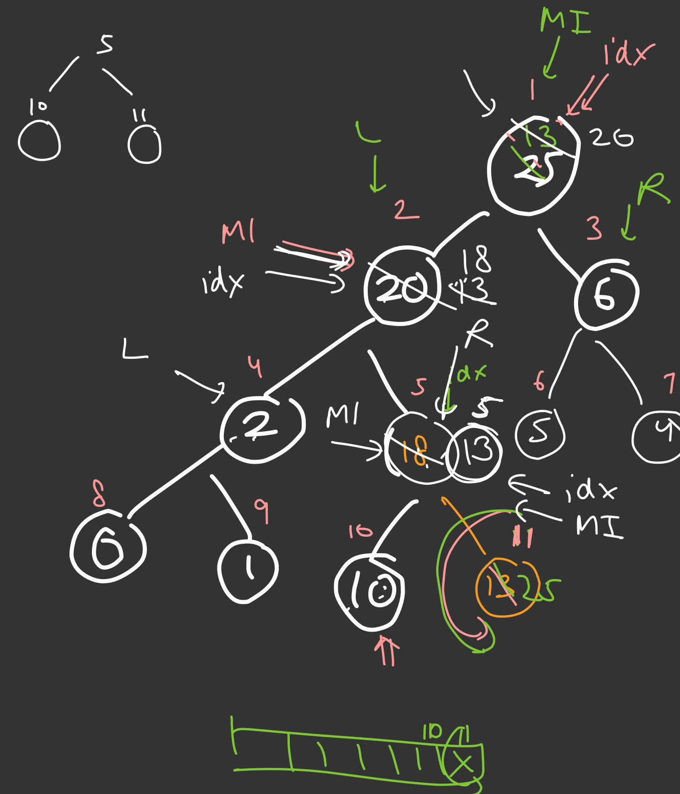
$right = 2 \cdot idx + 1$

$maxIdx = ?$



$size = 11$

$size = size - 1$
$= 10$



MI
idx
L
R
MI
idx
L
R
idx
MI
idx
MI

Break :)

10 45

Problem

Given lengths of N ropes, the cost of merging two ropes is equal to the sum of their lengths
Find the min cost to merge all the ropes in a single rope:

ropes = [1, 4, 2, 5]

5

7

12

3 times

2 times

1 time

Min cost ?

1 4

5 cost = 5 (1+4)

5 2 cost = 7 (1+4, 2)

7 5 cost = 12 (1+4+2, 5)

24

greedy Strategy

⇒ Should pick small len ropes first

↓ less total contribution

~~1~~, 4, ~~2~~, 5 ⟶ Min Ropes = (1, 2)

cost = 0 + 3 + 7 + 12

= 22

~~3~~, ~~4~~, 5, 7 ⟶ Min Ropes = (3, 4)

~~5~~, ~~7~~ ⟶ Min Ropes = (5, 7)

12 ⟶ Stop.

Use PQ class to build the Algo

5 mins

ropes [] = [1, 4, 2, 7, 8 --]

3

PriorityQueue pq = new PriorityQueue(); (natural ordering ↓ Min Heap)

```
for( rope : ropes)
        pq add (rope)

cost = 0

while ( pq size() > 1 ){

      R1 = pq.getMin() ; pq remove Min();
      R2 = pq.get Min(), pq remove Min(),

      cost = cost + (R1+R2);

      pq add (R1+R2);

}
print(cost)
```

poll() in Java

[1, 4, 2, 3, 7 . . . .]

(1, 2, 3, 4, 7 . . . . .)

$\boxed{3}$

(6, 8, 5, 9, 20, 17)

$\boxed{11} + \boxed{17}$

new data is correctly placed

O(log N) ← {insert}